Excel VBA 用グラフィックス・ライブラリの高速化

名誉教授　　木原 寛
情報政策課 技術専門職員 畑 篤

概要：先に報告した Excel VBA の AddShape 命令を利用したグラフィックス・ライブラリの各命令を、Windows API の GDI32.dll に含まれる関数を利用するよう書き直すことにより、最大約 7,000 倍の高速化を実現することができた。スクリーン座標系、ワールド座標系およびタートルグラフィックス用ライブラリを作成し公開した。

キーワード：VBA、Excel、プログラミング、グラフィックス、GDI32

１．はじめに

我々は先に、初心者向けのプログラミング学習環境としての Excel VBA での利用を念頭に置き、Excel VBA 用のワールド座標系グラフィックスおよびタートルグラフィックス・ライブラリを開発し報告した。[1] その際、Excel VBA の AddShape（図形作図）機能を利用して各作図ルーチンを作成した。そのため実行速度が遅いという欠点があったが、学習目的で簡単な作図を行うだけなので、とくに支障は無いと考えていた。ところが、最近になって、これらのグラフィックス・ライブラリを実用的な目的で使用している利用者も存在することがわかった。そこで、公開後に判明したその他の欠点の解消も図りつつ、Excel VBA 用のグラフィックス・ライブラリの高速化を検討することにした。

２．グラフィックス・ライブラリの概要

2.1 高速グラフィックス・ライブラリの作成

作図命令を高速化する方法を検討した結果、ゲームの開発などで用いられる Windows API の GDI32.dll に含まれる関数を利用できることがわかった。[2] そこで、GDI32 の関数を利用し、ワールド座標でのグラフィックス描画命令を標準モジュール内のプロシージャとして定義しライブラリ化した。（表１）AddShape 機能を利用したグラフィックス・ライブラリの場合に倣い、プロシージャの引数の並びは NEC PC-9801 の N_{88}-BASIC に準拠した形式とした。ラインスタイルや色の指定などに関する座標や主要なパラメータ変数は宣言部で定義し、大域変数としてプロシージャ間での値の引渡しを行うこととした。Visual Basic の定数については、VBA のヘルプの記述を参考にした。

2.2 グラフィックス・ライブラリの動作環境

Windows API を利用しているため、AddShape 機能を利用したライブラリの場合とは異なり、動作環境は Windows のみとなり、Mac では使用できない。

2.3 グラフィックス・ライブラリの改善点

前報[1]で報告した AddShape 機能を利用したライブラリでは、点の描画をサイズ１の箱や円を描く命令で代用していたため、描きむらが発生することがあったが、今回作成したライブラリでは本来の PointSet 命令を利用することができるため、そのような現象を回避することができる。図１に 3dxy 原子軌道関数の動径確率密度分布を描画した場合の比較を示す。また、図内にテキストを表示する簡単な機能を追加した。

各自のプログラムに座標変換機能が既に含まれている場合に合わせ、物理（スクリーン）座標系グラフィックス・ライブラリも用意した。

2.4 グラフィックス・ライブラリの利用法

ライブラリのソースプログラムおよびExcel VBAの標準モジュールにグラフィックス描画命令プロシージャを組み込んだファイルを配布している。[3] 利用者が、Visual Basic Editorで標準モジュールのModule１などにプログラムを

表１　ワールド座標によるグラフィックス描画プロシージャ

```
Sub InitializeGraphics()　　’グラフィックス利用の開始宣言
Sub SetViewPort(ViewLeft, ViewTop, ViewRight, ViewBottom)　　’ビューポートの指定
’描画ウィンドウのサイズの指定
Sub SetGraphicsWindow(WindowLeft, WindowTop, WindowRight, WindowBottom)
Sub Move(x, y)　　’移動
Sub DrawLineTo(x2, y2, Optional cLineRGB)　　’現在位置から直線を描く
Sub DrawLine(x1, y1, x2, y2, Optional cLineRGB)　　’直線を描く
Sub DrawPolyLine(x, y, n)　　’直線を連続して描く
Sub DrawRectangle(x1, y1, x2, y2, Optional cLineRGB)　　’矩形を描く
’塗りつぶした矩形を描く
Sub DrawRectangleFill(x1, y1, x2, y2, Optional cLineRGB, Optional cAreaRGB)
Sub DrawOval(x, y, rx, Optional ry, Optional cLineRGB)　　’楕円を描く
’塗りつぶした楕円を描く
Sub DrawOvalFill(x, y, rx, Optional ry, Optional cLineRGB, Optional cAreaRGB)
Sub PointSet(x, y, Optional cLineRGB)　　’点を打つ
Sub DrawText(x, y, St, Optional cTextRGB)　　’文字を表示する
Sub SetLineColor(lc)　　’線の色の指定
Sub SetLineStyle(ls)　　’線種の指定
Sub SetDashStyle(ds)　　’破線の種類の指定
Sub SetLineWidth(lw)　　’線の太さの指定
Sub gClear(Optional bc)　　’Viewport内を背景色で塗りつぶす
```

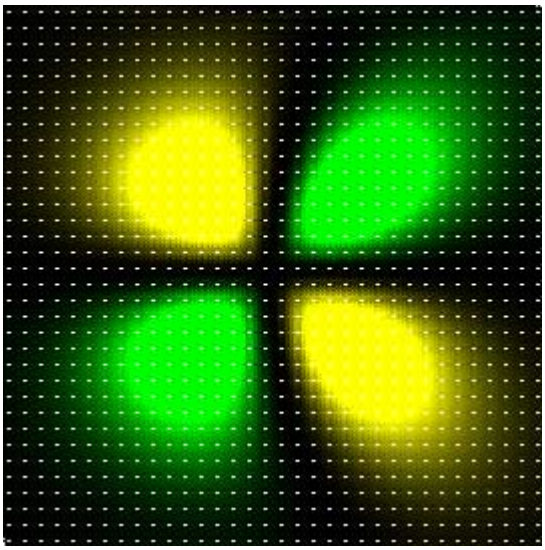

図１ａ　AddShape機能を利用したライブラリによる作図例

図１ｂ　GDI32 を利用したライブラリによる作図例

記述して使用する。詳しい利用法については、前報[1]およびWeb上で公開している情報やサンプルプログラムの内容を参考にしていただきたい。[3]

３．実行速度の比較

前報で報告したAddShape機能を利用したライブラリと今回作成したGDI32の関数を利用したライブラリによる作図速度を比較した結果を表２に示す。測定は、Macbook Air（CPU Intel Core i5-4250U 1.30GHz）のBootcamp上のWindows 8.1で実施した。

Line命令では、繰り返し回数によらずほぼ200倍以上の高速化が認められた。Polyline命令では100,000点の作図で7,000倍以上の高速化が見られた。その他の作図命令でも数十倍から数百倍の高速化が達成されている。

特筆すべきは、AddShape機能を利用したライブラリでは、描画オブジェクトの数が多くなると、ある段階から処理速度が急激に低下し、さらには「応答なし」となる現象が発生するのに対し、GDI32の関数を利用したライブラリでは、描画オブジェクトの数が増えても１命令当たりの速度がまったく低下せず、むしろ向上している点である。したがって、学習目的だけではなく実用的な目的での利用にも充分耐えうると期待される。

表２　作図命令の実行速度の比較

	回数	Autoshape	GDI32	速度比
Line	1,000	2.6	0.012	221
	10,000	38.5	0.156	246
	100,000	259.3	1.172	221
Polyline	10,000	39.3	0.009	4,596
	100,000	532.6	0.074	7,180
Pset	900	5.3	0.000	
	10,000	164.1	0.063	2,626
	90,000	×	0.516	
	250,000	ー	1.637	
	1,000,000	ー	6.016	
Rectangle	1,000	6.4	0.031	204
	10,000	213.1	0.141	1,515
	100,000	×	1.215	
RectangleFill	1,000	6.6	0.234	28
	10,000	212.8	2.000	106
	100,000	ー	19.895	
Oval	1,000	6.6	0.047	141
	10,000	215.9	0.379	570
	100,000	ー	2.734	
OvalFill	1,000	7.6	0.109	69
	10,000	222.4	0.824	270
	100,000	ー	7.852	

×：応答なしとなる　　ー：応答なしとなる可能性または長時間を要するために未実施

参考文献及び注

1）木原　寛、「Excel VBA へのグラフィックス描画命令の実装」，富山大学総合情報基盤センター広報，Vol.4，p.35（2007）
2）近田伸矢 他,「アクションゲーム作成入門」，インプレスジャパン，第 8 章　（2009）
3）http://katakago.sakura.ne.jp/pgm/vba/index.html に、Excel VBA 用グラフィックス・ライブラリとサンプルプログラムを掲載している。